МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра параллельных вычислений

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Абрамушкиной Екатерины Сергеевны

Тема работы:

РАЗРАБОТКА И РЕАЛИЗАЦИЯ ПОДСИСТЕМЫ ПРОФИЛИРОВАНИЯ ДЛЯ СИСТЕМЫ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ LUNA

«К защите допущена»

Заведующий кафедрой ПВ ФИТ НГУ,
д.т.н., профессор
Малышкин В.Э. / ……
31 мая 2023 г.

Руководитель ВКР

к.т.н., доцент
доцент каф. ПВ ФИТ НГУ
Маркова В.П. / ……
20 мая 2023 г.

Соруководитель ВКР

ст. преп. каф. ПВ ФИТ НГУ
Киреев С.Е. / ……
20 мая 2023 г.

Новосибирск, 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра параллельных вычислений

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ
Зав. кафедрой ПВ ФИТ НГУ
Малышкин В.Э.
……
07 ноября 2022 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студент*ке* Абрамушкиной Екатерине Сергеевне, группы 19202.

Тема Разработка и реализация подсистемы профилирования для системы фрагментированного программирования LuNA

утверждена распоряжением проректора по учебной работе от 07.11.2022 № 0330.

Срок сдачи студентом готовой работы: 20 мая 2023 г.

Исходные данные (или цель работы): разработать и реализовать подсистему профилирования в системе фрагментированного программирования LuNA.

Структурные части работы: обзор и анализ предметной области, постановка задачи, формулирование требований к подсистеме, разработка и реализация прототипа подсистемы, тестирование.

Руководитель ВКР

к.т.н., доцент,
доцент каф. ПВ ФИТ НГУ
Маркова В.П. / ……
07 ноября 2022 г.

Задание приняла к исполнению

Абрамушкина Е.С. / ……
07 ноября 2022 г.

Соруководитель ВКР

ст. преп. каф. ПВ ФИТ НГУ
Киреев С.Е. / ……
07 ноября 2022 г.

СОДЕРЖАНИЕ

Введение....4
1 Термины и определения....7
1.1 Термины, связанные с технологией фрагментированного программирования....7
1.2 Термины, связанные с профилированием программ....7
2 Обзор средств профилирования параллельных программ....10
2.1 Обзор средств профилирования параллельных MPI-программ....10
2.2 Обзор существующих средств профилирования в LuNA....16
2.3 Результаты обзора....18
3 Проектирование архитектуры подсистемы профилирования....20
3.1 Функциональные требования к подсистеме профилирования....20
3.2 Нефункциональные требования к подсистеме профилирования....23
3.3 Описание предлагаемого решения....25
3.4 Анализ предлагаемого решения....28
4 Реализация прототипа подсистемы профилирования....32
4.1 Описание деталей реализации прототипа подсистемы профилирования....33
4.2 Реализация алгоритма синхронизации часов....34
5 Тестирование прототипа подсистемы профилирования....36
5.1 Измерение доли накладных расходов на работу прототипа подсистемы....36
5.2 Тестирование функциональности прототипа подсистемы....38
5.3 Итоги тестирования....39
Заключение....41
Список использованных источников и литературы....43
Приложение А....47
Приложение Б....53

ВВЕДЕНИЕ

В современном мире в различных научных сферах часто встречаются задачи, для решения которых нужно применять методы численного моделирования. Численное моделирование – это методология, которая заключается в построении математической модели изучаемой системы и дальнейшем исследовании модели с использованием численных методов, реализованных на компьютере. Численное моделирование позволяет изучать поведение системы в различных условиях, а также предоставляет возможность получать более точные и детальные результаты экспериментов, проведение которых в реальности связано с техническими, экономическими трудностями. Таким образом, данный подход может сократить затраты на исследования, поэтому он широко используется для исследования химических, биологических и физических процессов, а также экологических и экономических систем.

Реализация модели на компьютере является трудным процессом, поскольку она требует от специалиста компетенций не только в предметной области, но и в области параллельного программирования и современных вычислительных систем. Это связано с тем, что решение задач численного моделирования зачастую подразумевает большие объемы вычислений. Обеспечить вычислительные мощности могут только высокопроизводительные кластерные компьютерные системы, для использования которых возникает потребность в создании параллельных распределенных программ, реализующих заданный численный алгоритм.

Одним из направлений работы в ИВМиМГ СО РАН является создание методов и средств параллельной реализации больших численных моделей на суперкомпьютерах. В связи с этим в ИВМиМГ СО РАН разрабатываются технология фрагментированного программирования [1, 2] и реализующие ее язык и система программирования LuNA [3, 4]. Система программирования LuNA является системой автоматического конструирования параллельных программ. Такие системы помогают абстрагировать программиста от части проблем параллельного программирования, таких как динамическая балансировка нагрузки, настройка под имеющиеся аппаратные ресурсы вычислительных узлов и коммуникационные взаимодействия потоков исполнения. Таким образом, написание параллельной программы с использованием системы программирования LuNA становится задачей менее трудоемкой, чем создание подобной программы стандартными средствами (например, MPI [5]).

Важно, чтобы реализованный параллельный алгоритм имел высокую производительность, то есть программа должна эффективно использовать ресурсы системы для решения поставленной задачи. Обнаружение причин низкой эффективности параллельной программы является сложной задачей, поскольку на это влияет большое

количество факторов. Одним из способов анализа производительности программы является профилирование, то есть сбор характеристик исполнения программы, которые впоследствии используются для исследования поведения программы. В качестве примеров таких характеристик можно привести время исполнения программы, использование процессора, памяти, сетевой подсистемы, коммуникаций между потоками исполнения.

В системе программирования LuNA профилирование исполняемых фрагментированных программ используется различными подсистемами, общей целью которых является повышение производительности системы LuNA. Однако на данный момент средства профилирования в системе LuNA представляют собой разрозненный набор инструментов, каждый из которых был разработан для конкретной задачи и имеет ограниченную область применения. Несмотря на то, что профили, получаемые с помощью различных инструментов, содержат схожую информацию о состоянии системы, при разработке этих инструментов не учитывалась возможность переиспользования уже существующей в системе функциональности. Как следствие, реализации инструментов частично дублируют друг друга, форматы представления профилей несовместимы, хотя профили и содержат схожую информацию. Задача также осложняется тем, что каждый из инструментов был задуман, разработан и использовался одним и тем же человеком, что привело к отсутствию документации. Все эти проблемы отрицательно отражаются на качестве кода системы, усложняют его понимание, модификацию и поддержку.

Профилирование занимает важное место в процессе анализа и оптимизации производительности системы программирования LuNA в настоящее время и будет использоваться в будущем при создании новых методов повышения производительности системы. В связи с этим в системе программирования LuNA актуальной задачей является разработка подсистемы профилирования, которая не только позволит сохранить существующую функциональность, но и предоставит возможности для ее расширения, а также станет фундаментом для дальнейшего развития инструментов профилирования в системе, обеспечив совместимость между ними.

Цель работы – разработать и реализовать подсистему профилирования в системе фрагментированного программирования LuNA.

Для достижения данной цели были поставлены следующие задачи:

- изучить существующие в системе средства для профилирования фрагментированных программ;
- составить и проанализировать список требований, которым должна удовлетворять подсистема профилирования;

- спроектировать архитектуру подсистемы профилирования на основе поставленных требований;
- реализовать и протестировать прототип подсистемы профилирования.

Научная новизна заключается в разработке подсистемы профилирования в системе фрагментированного программирования LuNA, которая позволит объединить существующие инструменты профилирования и предоставит возможности для расширения функциональности средств профилирования в системе LuNA.

Практическая ценность работы состоит в том, что разработанная подсистема позволит унифицировать процесс сбора информации о событиях, происходящих в процессе исполнения фрагментированных программ. Подсистема предоставит единый интерфейс для работы с собранными данными, для использования которого в работе также предложен универсальный и расширяемый формат профиля.

Работа состоит из пяти глав и заключения. В первой главе описаны используемые в работе термины и их определения. Во второй главе рассмотрены существующие способы и средства профилирования параллельных программ и их применимость для сбора данных об исполняемых программах в системе программирования LuNA. В третьей главе сформулированы функциональные и нефункциональные требования, которым должна удовлетворять разрабатываемая подсистема профилирования, а также описаны принятые архитектурные решения, их анализ и обоснование. В четвертой главе приведены детали реализации подсистемы профилирования. В пятой главе представлены результаты тестирования прототипа подсистемы. В заключении описаны результаты работы.

1 Термины и определения

Для начала необходимо ввести термины и определения, которые будут использоваться в работе.

1.1 Термины, связанные с технологией фрагментированного программирования

Система фрагментированного программирования LuNA – это система программирования, использующая крупноблочный (крупнозернистый) параллелизм для распараллеливания задач. Это означает, что распараллеливание происходит путем распределения крупных задач по вычислительным узлам мультикомпьютера [6].

Система программирования LuNA реализует технологию фрагментированного программирования, которая поддерживает процесс сборки целой программы из фрагментов вычислений и ее исполнение. *Фрагмент вычислений* – это независимая единица программы, которая содержит описание входных/выходных переменных и кода фрагмента [2]. Фрагменты вычислений принимают на вход и вырабатывают *фрагменты данных* – неизменяемые блоки данных заданного размера.

Фрагментированная программа на языке LuNA представляет собой компактную запись потенциально бесконечного множества фрагментов, реализующих заданный параллельный алгоритм. Таким образом, фрагментированная программа определяется как множества переменных (фрагментов данных) и фрагментов вычислений. При записи программы используются операторы языка LuNA, такие как циклы **for** и **while**, условные операторы **if**, подпрограммы.

Исполнение фрагментированных программ в системе LuNA обеспечивает *исполнительная система*, которая выполняет операции по созданию, перемещению фрагментов вычислений и фрагментов данных. Исполнение фрагментов вычислений происходит по готовности входных фрагментов данных, после исполнения фрагмента вычислений все его выходные фрагменты данных получают свои значения. Порядок исполнения фрагментов вычислений в системе не является детерминированным и определен лишь информационными зависимостями между ними [2]. Это означает, что, если фрагмент вычислений CF1 потребляет фрагмент данных, который вырабатывается после исполнения фрагмента вычислений CF2, то исполнение CF1 не начнется, пока не исполнится CF2.

1.2 Термины, связанные с профилированием программ

Профилирование – это сбор характеристик исполнения программы, которые впоследствии используются для исследования поведения программы. Чаще всего профилирование применяется для анализа производительности программного обеспечения, так как оно позволяет определить, какие операции выполняются наиболее часто, какие

участки кода потребляют наибольшее количество ресурсов, например, процессорного времени или памяти.

Профилирование может быть статическим и динамическим. *Статическое профилирование* выполняется без запуска программы. Такой метод профилирования подразумевает построение некоторой математической модели по исходному коду и последующий анализ этой модели для определения поведения программы. *Динамическое профилирование* отличается от статического тем, что сбор характеристик происходит во время исполнения программы. Динамическое профилирование является более распространенным, поскольку оно позволяет получить более точное представление о производительности программы.

Динамическое профилирование требует изменения исходного кода программы, поэтому для этого метода профилирования часто используются специальные инструменты и библиотеки. Инструмент, выполняющий профилирование программы, называется *профилировщиком*. Внедрение в исходный код программы инструкций для вызова профилирующих инструментов и сбора профиля называется *инструментированием*.

Для реализации средств профилирования используются различные способы, которые можно разбить на две группы: детерминированные и статистические.

При *детерминированном профилировании* инструкции для профилирования внедряются непосредственно в код и выполняются каждый раз, когда исполнение достигает определенных точек. Они исполняются ровно столько раз, сколько исполняется инструментированный участок кода, и дают точную информацию. При использовании данного способа нужно иметь в виду, что частота их исполнения прямо пропорциональна частоте исполнения основного кода. Наиболее чувствительными к инструментированию оказываются часто исполняемые участки кода: часто вызываемые методы и короткие циклы с большим количеством итераций. Такие участки чаще всего исследуются с целью оптимизации, поэтому при сборе информации о них необходимо учитывать, что в таких местах вставка инструкций профилировщика сильнее всего сказывается на скорости работы программы.

Другим способом является *статистическое профилирование* (также встречаются названия «выборочное профилирование» или «сэмплинг» – от англ. *sampling profiling*). В этом случае с некоторой периодичностью (через некоторый фиксированный интервал времени или один раз за некоторое фиксированное число выполнений данного участка кода) делаются выборки информации о состоянии исполняемой программы, и затем на основе этой выборочной информации принимаются решения. Информация об исполнении программы

при использовании данного способа не является точной, но, при правильном подборе частоты выборок, ее достоверность может достигать 95% и выше [7].

И детерминированное, и выборочное профилирование подразумевают промежуточную обработку информации для сбора профиля во время исполнения программы. Им можно противопоставить такой способ, как *«посмертное» профилирование* (от англ. *post mortem profiling*). Этот способ основан на *трассировке* – записи без промежуточной обработки информации о текущем состоянии программы во время ее исполнения. Для трассировки используются специальные инструменты, называемые *трассировщиками*, которые регистрируют все события в программе и генерируют трассировочные файлы (трассы) с данными об этих событиях. Трасса содержит подробную информацию о том, что происходило в процессе исполнения программы, и эта информация, в отличие от профиля, никак не обрабатывается и не анализируется во время исполнения программы.

Трассировку также иногда сравнивают с логированием. *Логирование* – это процесс сбора информации о работе системы в специальные файлы, называемые *логами*. Несмотря на внешнюю схожесть, между данными терминами есть различие. Во время трассировки записывается информация о событиях, происходящих в системе на низком уровне. Их количество может исчисляться сотнями, тысячами. В логи же записывается информация о высокоуровневых событиях, которые случаются гораздо реже: например, вход пользователей в систему, критические ошибки в работе приложения. Логирование также используется для исследования поведения программы, но чаще применяется для поиска причин возникновения ошибок в системе и практически не имеет отношения к исследованию производительности программ.

2 Обзор средств профилирования параллельных программ

Прежде всего необходимо рассмотреть существующие средства для профилирования параллельных программ и функциональность, которую они предоставляют пользователю, чтобы проанализировать, могут ли данные средства быть использованы для анализа фрагментированных программ, исполняемых в системе программирования LuNA.

2.1 Обзор средств профилирования параллельных MPI-программ

2.1.1 MPE

MPE (Multi-Processing Environment) – это средство для профилирования параллельных программ, которое разработано в Аргоннской национальной лаборатории США. Алгоритм работы основан на посмертном профилировании. Библиотека свободно распространяется вместе с пакетом MPICH [8], но может также использоваться и другими реализациями MPI [9, 10].

Библиотека предоставляет полный набор инструментов для анализа производительности MPI-программ, который содержит:

- подпрограммы для сбора информации о поведении программы и их обработки;
- скрипты для компиляции программы с поддержкой MPE, утилиты для конвертации трассировочных файлов в разные форматы, для их фильтрации и распечатки;
- визуализатор трассировочных файлов Jumpshot для отображения результатов профилирования.

Запись трасс исполнения в MPE может работать в двух режимах: автоматическом и ручном. В процессе исполнения программы в режиме автоматического профилирования для каждой вызываемой MPI-функции замеряется время перед вызовом и после него. В ручном режиме пользователь описывает модель программы в виде конечного автомата, записи подлежат события, которые соответствуют моменту смены состояния автомата.

С помощью утилиты Jumpshot можно отобразить статистику различных событий, детализацию профиля в заданной временной области, в которой представлены моменты начала или конца работы MPI-подпрограммы, моменты смены состояния автомата, пути пересылки данных между узлами. Графический интерфейс утилиты представлен на рисунках 1 и 2. Временные интервалы, которые соответствуют работе MPI-функций в MPI-процессах, изображены на рисунке 1 при помощи разноцветных прямоугольников. На рисунке 2 показаны условные обозначения для отображения результатов профилирования.

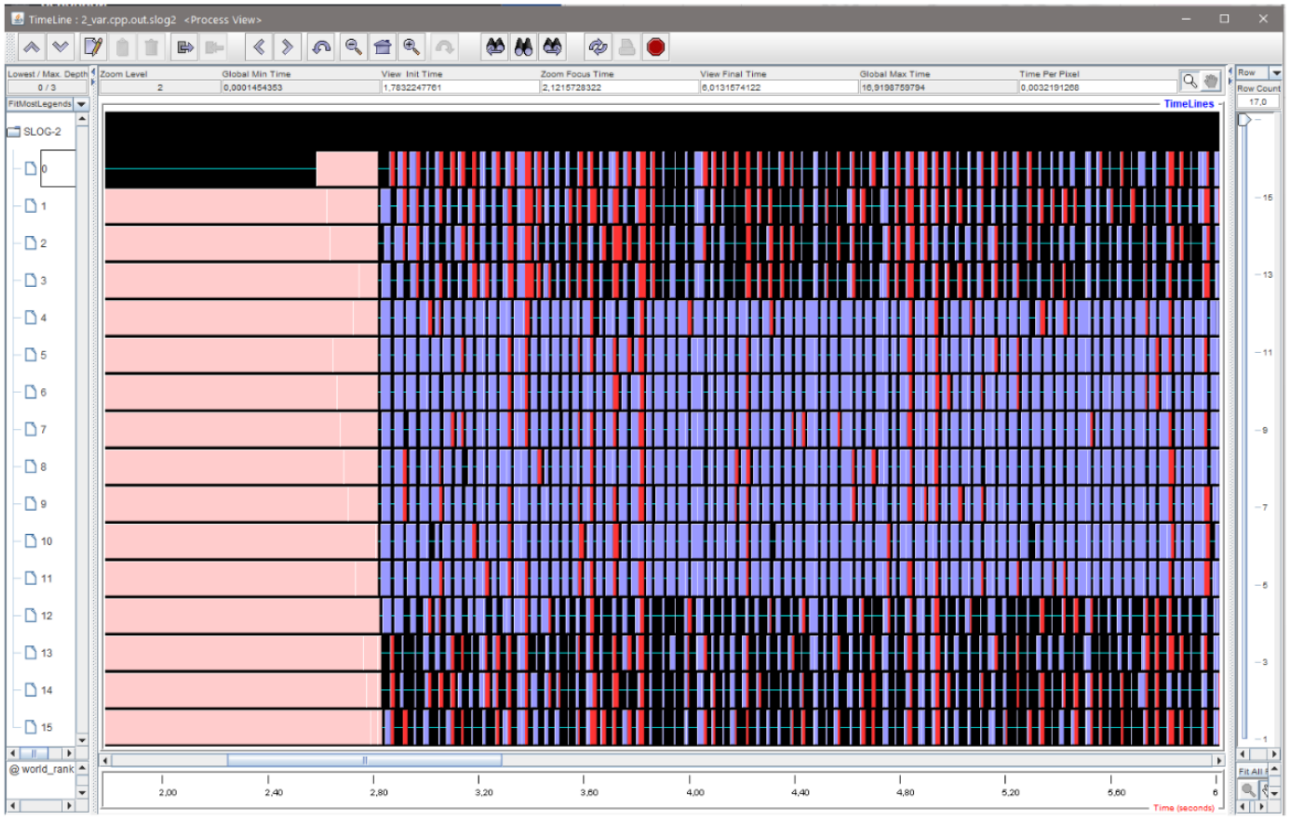

Рисунок 1 – Графический интерфейс утилиты Jumpshot

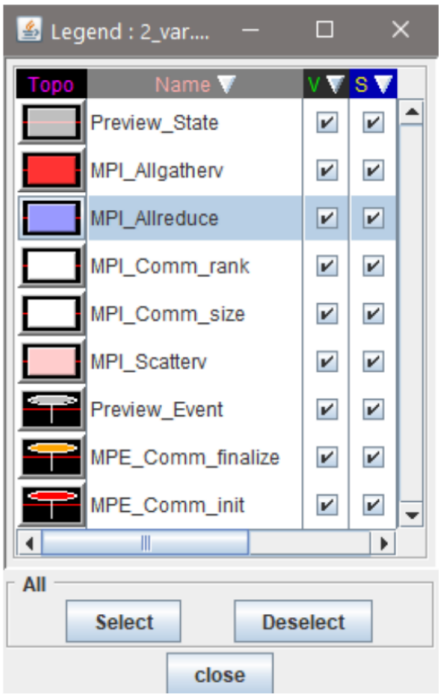

Рисунок 2 – Условные обозначения утилиты Jumpshot

2.1.2 Extrae, Paraver

Extrae и Paraver – это средства для профилирования MPI-программ, которые входят в свободно распространяемый набор инструментов BSC Performance Analysis Tools, разрабатываемый в Суперкомпьютерном центре в Барселоне совместно с компанией Intel [11]. Эти два инструмента предназначены для совместного использования. Алгоритм работы

также основан на посмертном профилировании. Extrae – это пакет для инструментирования MPI-программ, который используется для записи трассы, а Paraver ее визуализирует [12, 13]. Поддерживается несколько режимов визуализации при помощи временных диаграмм:

- отображение поведения MPI-программы аналогично Jumpshot;
- отображение производительности в флопсах для заданного процесса;
- отображение количества инструкций в секунду (IPS) на вычислителе для заданного процесса.

2.1.3 Pablo Performance Analysis Toolkit

Pablo Performance Analysis Toolkit – это некоммерческий пакет для анализа производительности программ, разработанный в кампусе в Урбане-Шампейне Иллинойского университета. Пакет представляет собой графическую среду для инструментирования исходного кода и динамического профилирования приложений.

Пакет включает в себя два основных компонента:

- набор средств для инструментирования кода и трассировки программ;
- набор средств для анализа производительности на основе трассы с использованием специального мета-формата (SDDF), который связывает два набора средств между собой [14].

Пакет состоит из библиотеки для записи событий с временными метками и расширений для работы с MPI-программами, которые предоставляют возможность записывать информацию о вызовах MPI-функций, анализировать операции ввода и вывода MPI-функций. Пользователь может выбирать между режимом детальной трассировки, в котором записывается каждая MPI-операция ввода-вывода, или режимом трассировки, который кратко резюмирует данные о производительности для каждого типа MPI-функции ввода-вывода [9, 15].

Трасса программы записывается в формате SDDF (Self-Describing Data Format). Пакет содержит набор утилит для работы с файлами формата SDDF и конвертеры для преобразования трасс в другие форматы. Также в набор входит инструмент Performance Capture Facility (PCF), который позволяет не только записывать трассу, но и отслеживать производительность приложения в реальном времени. PCF можно также использовать для многопоточных MPI-программ. Для визуализации используется SvPablo – средство, предоставляющее графический пользовательский интерфейс для инструментирования кода и визуализации трасс.

2.1.4 Vampir, VampirTrace

VampirTrace – это бесплатная библиотека с открытым исходным кодом для трассировки MPI-программ и многопоточных программ, разработанная в Центре

высокопроизводительных вычислений в Дрезденском техническом университете [16]. Vampir – это коммерческий фреймворк для визуализации и анализа трасс, полученных при помощи VampirTrace и некоторых других инструментов (например, TAU и KOJAK) [17].

VampirTrace имеет несколько режимов инструментирования кода: ручной, автоматический при помощи компилятора, бинарный. Трасса, собираемая при помощи VampirTrace, содержит детальную информацию о событиях и позволяет пользователю исследовать поведение программы. Трасса записывается в формате OTF (Open Trace Format) и содержит информацию о коммуникациях типа «точка-точка», коллективных коммуникациях, MPI-функциях ввода и вывода, пользовательских подпрограммах [18].

Графический интерфейс инструмента Vampir представлен на рисунке 3. Временная диаграмма в левой части рисунка отображает информацию о событиях для процессов. Справа в верхней части расположена статистика о суммарном времени для каждой группы функций, в нижней части – легенда диаграммы.

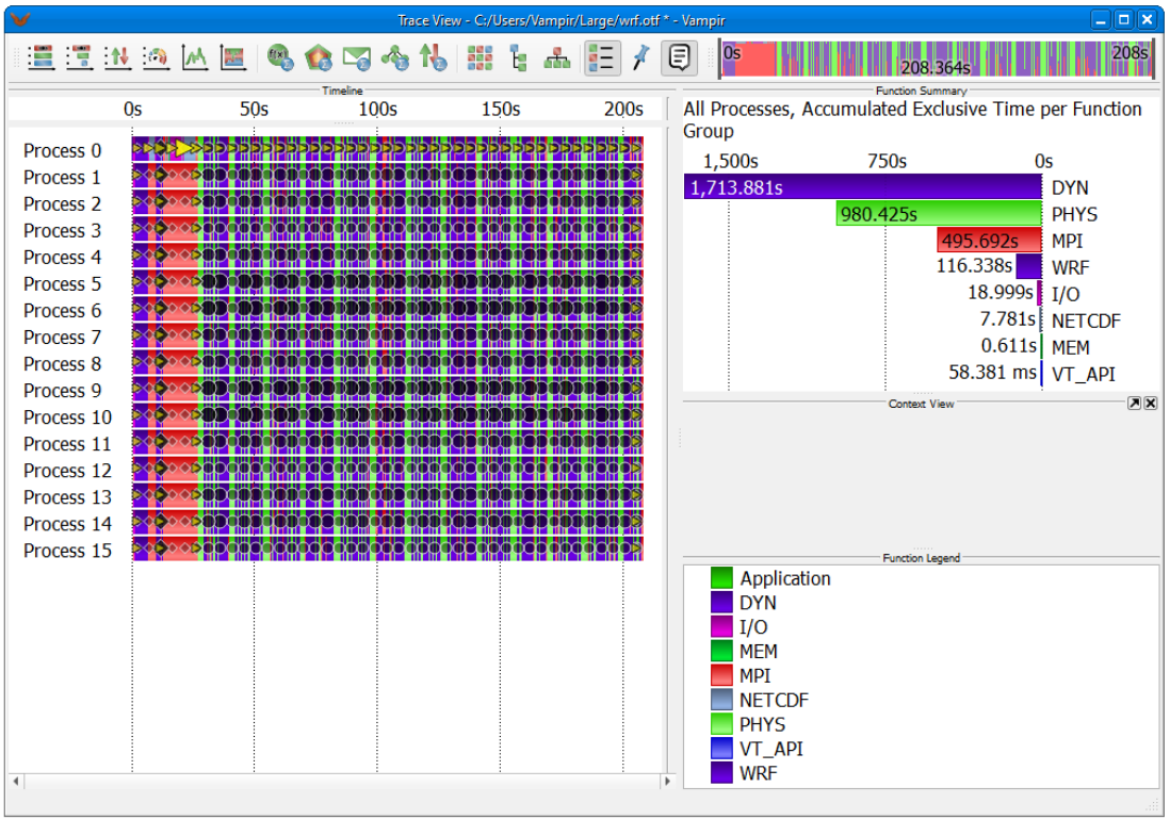

Рисунок 3 – Графический интерфейс визуализатора трасс Vampir

2.1.5 Intel Trace Analyzer and Collector

Intel Trace Analyzer and Collector – это пакет для анализа производительности параллельных приложений. Пакет состоит из двух средств: Intel Trace Collector и Intel Trace Analyzer. Intel Trace Collector – это инструмент, предназначенный для трассировки параллельных приложений, использующих MPI, приложений, работающих с общей памятью, а также приложений, не использующих MPI. Intel Trace Analyzer – это инструмент для

анализа и визуализации результатов трассировки приложений. Данный пакет также можно использовать для многопоточных MPI-программ.

Пользователю предоставляются следующие возможности [19]:

- Обнаружение взаимных блокировок процессов, проверка корректности вызовов MPI-подпрограмм: аргументов, типов данных, буферов, коммуникаторов.
- Анализ времени работы отдельных подпрограмм.
- Сбор статистики по событиям, оценка сбалансированности загрузки вычислителя.
- Выделение участков программы, активно использующих подпрограммы для коммуникации между узлами, выявление в этих участках шаблонов взаимодействия процессов, негативно сказывающихся на производительности.
- Симуляция исполнения программы с нулевыми задержками и бесконечной пропускной способностью в сетевой подсистеме. Сравнивая полученный в результате симуляции профиль и первоначальный, Intel Trace Analyzer может указать на сетевую подсистему как на одну из причин низкой производительности.

2.1.6 Scalasca (KOJAK)

Система Scalasca [20] (и ее предшественник KOJAK [21]) используется для анализа производительности программ с использованием MPI, OpenMP, POSIX threads [21]. Это средство профилирования программ разрабатывается Юлихским исследовательским центром совместно с Техническим университетом Дармштадта. Система предоставляет пользователю возможность автоматически находить причины неэффективного исполнения программ на высоком уровне абстракции. Пользователь также может выбрать один из двух режимов работы системы:

- Сбор профиля – краткого отчета об основных метриках производительности.
- Сбор трассы – детальной информации о событиях в исполняемой программе, на основе которой затем можно составить профиль или временную диаграмму.

В режиме сбора трассы для каждого процесса генерируется отдельный трассировочный файл. После завершения программы все файлы обрабатываются системой параллельно, задействуя столько же процессов, сколько было использовано при выполнении исследуемой программы. Во время анализа Scalasca создает и исследует более высокоуровневое промежуточное представление программы. Система ищет заранее известные шаблоны взаимодействия процессов, характерные для параллельных программ, которые могут снижать производительность программы, разделяет их по категориям, упорядочивает по важности и связывает с исходным кодом программы [22]. Примерами обнаруживаемых шаблонов могут быть неправильный порядок приема и передачи

информационных зависимостей, поздний вызов блокирующих подпрограмм широковещательной коммуникации и т.д.

2.1.7 TAU (Tuning and Analysis Utilities)

TAU представляет собой свободно распространяемый набор средств для измерения производительности, созданный в Орегонском университете в США. Этот инструмент можно использовать для профилирования и для трассировки программ.

Трассировка в TAU выполняется при помощи инструментирования, для этого программа представляется в виде последовательности событий. Инструментирование может производиться пользователем вручную, для этого в TAU предусмотрено пять различных интерфейсов, влияющих на используемую технику профилирования. Также код может быть инструментирован автоматически при помощи препроцессора, компилятора или специальной библиотеки, которая заменяет вызов подпрограммы на его инструментированную версию [23].

При профилировании TAU может работать в двух режимах: PProf и ParaProf. В режиме PProf средство анализирует код и составляет список функций в порядке возрастания времени потребления. ParaProf показывает ту же информацию, что и PProf, но отображает ее графически с помощью диаграмм и трехмерных изображений. С помощью TAU можно собрать полные пути вызова, чтобы получить лучшее представление о контексте каждого вызова функции.

TAU не содержит встроенных средств для визуализации трасс, для этого предназначен отдельный инструмент под названием Racy [9, 24]. Также в TAU предусмотрена конвертация трассировочных файлов из его собственного формата в другие форматы, поэтому для визуализации трассы можно использовать другие профилировщики. Например, поддерживаются такие форматы, как SLOG-2 (Jumpshot), EPILOG (KOJAK), OTF (VampirTrace).

2.1.8 HPCToolkit

HPCToolkit – это набор инструментов для измерения и анализа производительности параллельных программ. Он является частью проекта Exascale Computing Project (ECP), который разрабатывается двумя организациями Министерства энергетики США (US Department of Energy, DOE) – Управлением науки (the Office of Science, DOE-SC) и Национальным управлением ядерной безопасности (National Nuclear Security Administration, NNSA) [25].

HPCToolkit поддерживает работу с последовательными программами, с параллельными программами, использующими MPI, OpenMP, POSIX Threads, а также гибридными программами – использующими и MPI, и многопоточность.

Этот набор инструментов основан на статистическом профилировании. С помощью сэмплирования таймеров и аппаратных счетчиков процессора, HPCToolkit получает точные характеристики использования процессора, потребления ресурсов при исполнении программы и соотносит их с полным контекстом вызова, к которому они относятся [25].

2.2 Обзор существующих средств профилирования в LuNA

В рамках проекта по разработке системы программирования LuNA есть множество подпроектов, использующих профиль исполняемой фрагментированной программы. Профилировочная информация используется для автоматической балансировки нагрузки в системе на основе трасс [26], для оценки эффективности фрагментации алгоритма [27], для обнаружения причин зависания программ [28], для динамической балансировки вычислительной нагрузки при воспроизведении трасс [29].

2.2.1 Набор макросов для трассировки фрагментированных программ

Система программирования LuNA имеет набор макросов для вывода информации во время исполнения фрагментированных программ. Он включает в себя макросы, предназначенные для форматированного вывода информации в стандартный поток вывода ошибок. Помимо этого, в наборе также есть макросы TRACE и LOG, используемые для трассировки и логирования исполняемой программы. Макрос TRACE принимает в качестве аргумента текст сообщения и выводит его в стандартный поток вывода. Макрос LOG также принимает текст сообщения, но выводит сообщение в текстовый файл в специальном формате, дополняя его сопровождающей информацией: номером процесса, временной меткой в секундах с момента начала исполнения программы, а также названием файла и номером строки, где вызывается макрос. Запись о событии в логе имеет следующий формат:

Номер процесса | *Время* | *Информация о событии* | *Исходный файл:Строка*

2.2.2 WorkflowTracer

Другой инструмент для трассировки фрагментированных программ, исполняемых в системе LuNA, представлен классом WorkflowTracer [28]. Для создания объекта требуется передать в конструктор класса WorkflowTracer номер узла, который будет использован в названии трассировочного файла, а также размеры буферов отложенной записи в байтах для фрагментов вычислений и фрагментов данных.

WorkflowTracer был разработан для подсистемы обнаружения семантических ошибок во фрагментированных программах, поэтому он имеет несколько специальных методов. Методы *cf_created* и *cf_destroyed* используются для отслеживания фрагментов вычислений, существующих в исполнительной системе. Также трассировщик имеет метод *cfs_remaining*, который возвращает количество созданных фрагментов вычислений, которые еще не были уничтожены. Этот метод используется после остановки системы для проверки наличия

зависших фрагментов вычислений, то есть таких фрагментов вычислений, выполнение которых по тем или иным причинам не может завершиться в рамках нормальной работы исполнительной системы.

Для анализа трассировочных файлов, создаваемых WorkflowTracer, была разработана утилита *luna_trace*, обладающая следующими функциональными возможностями:

- автоматическая агрегация файлов трасс со всех узлов;
- предоставление информации о зависших фрагментах вычислений;
- предоставление информации о неинициализированных фрагментах данных;
- поиск повторно инициализированных фрагментов данных.

2.2.3 LuNA Evlog, LuNA Evlog Analyzer

В более ранних версиях системы LuNA для логирования фрагментированных программ использовался модуль Evlog [30]. Он позволял получать информацию о следующих событиях:

- начало работы фрагмента вычислений;
- окончание работы фрагмента вычислений;
- создание фрагмента данных;
- удаление фрагмента данных;
- начало отправки фрагмента данных;
- окончание приема фрагмента данных;
- отсутствие задач у потока исполнения;
- появление новых задач у потока исполнения.

Для логирования на стадии компиляции LuNA-программы выполнялось инструментирование: код на языке C++ дополняется вызовами методов класса Evlog. Помимо этого, классы CF и DF, представляющие фрагменты вычислений и фрагменты данных в системе LuNA, были дополнены следующей информацией:

- для фрагмента вычислений – название фрагмента вычислений, название пользовательской функции, исполняемой в данном фрагменте, тип фрагмента вычислений;
- для фрагмента данных – название фрагмента данных, номер родительского фрагмента вычислений.

LuNA Evlog Analyzer [31] – это инструмент, разработанный для анализа логов фрагментированных программ, собранных при помощи инструмента Evlog и содержащих информацию о событиях, происходивших во время исполнения программы.

Данный инструмент позволяет получить такие характеристики, как:

- объем оперативной памяти, потребляемой фрагментами данных;

- объем фрагментов данных, переданных по сети;
- количество одновременно исполняющихся фрагментов вычислений.

Для всех вышеперечисленных характеристики можно получить как интегральные значения, так и их зависимости от времени. Примеры извлекаемых характеристик: объем оперативной памяти, занимаемой фрагментами данных, объем передаваемых по сети фрагментов данных, темп передачи фрагментов данных или фрагментов вычислений по сети, количество фрагментов вычислений, готовых к выполнению, количество одновременно выполняющихся фрагментов вычислений.

2.3 Результаты обзора

На основе выполненного обзора можно сделать вывод о том, что существует большое количество средств, применяемых для профилирования параллельных программ, каждое из которых обладает специальными возможностями в зависимости от решаемой задачи. Однако, ни один из представленных проектов не смог бы решить проблему профилирования в системе программирования LuNA полностью.

Все инструменты, рассмотренные в подразделе 2.1, предоставляют пользователю низкоуровневую информацию о работе программы. Они позволяют получать данные об исполняемой программе в терминах операционной системы – процессах, потоках, – и терминах языка программирования низкого уровня: переменных, операторах и функциях. Такую информацию трудно соотнести с фрагментами данных и фрагментами вычислений в фрагментированных программах, поскольку они отличаются от программ, написанных на языках Fortran, C, C++, более высоким уровнем абстракции. Установление соответствия между множеством потоков и процессов каждого вычислительного узла и множеством фрагментов вычислений фрагментированной программы будет трудной задачей, поскольку исполнение фрагментов вычислений абстрагировано от процессов и потоков. Определение местоположений фрагментов данных (как временных, так и положений в оперативной памяти), которые могут пересылаться между узлами и процессами вычислительной системы, также усложняет задачу анализа фрагментированной программы. Таким образом, использовать рассмотренные средства профилирования для анализа производительности фрагментированных программ не представляется возможным, однако функциональность данных инструментов может служить ориентиром и источником идей для развития функциональности подсистемы профилирования в системе LuNA в будущем.

Инструменты для сбора и анализа трассы, рассмотренные в подразделе 2.2, позволяют получать информацию об исполнении фрагментированной программы в терминах фрагментов вычислений и фрагментов данных. Однако на данный момент средства

профилирования в системе LuNA представляют собой разрозненный набор инструментов и имеют ряд проблем.

Одна из главных проблем заключается в том, что в системе LuNA есть несколько средств для трассировки фрагментированных программ. Из обзора видно, что эти средства имеют схожий список обрабатываемых событий, что означает, что трассировочные файлы содержат схожую информацию в разных форматах. Это также подразумевает, что реализации данных инструментов частично дублируют друг друга. Дублирование кода увеличивает размер исходного кода системы, усложняет его понимание и создает трудности при внесении изменений.

Другой важной проблемой является отсутствие возможностей для расширения существующих средств профилирования, поскольку каждый инструмент был разработан для конкретной задачи и имеет ограниченную область применения. Система программирования LuNA имеет множество задач, связанных с отладкой и оптимизацией фрагментированных программ, для которых в будущем будет необходимо расширять возможности профилирующих средств. Для решения различных задач могут потребоваться и другие способы профилирования, в то время как все существующие инструменты основаны на посмертном профилировании.

По результатам обзора можно сделать вывод, что в системе программирования LuNA в настоящее время актуальной задачей является разработка универсальной подсистемы профилирования, которая решит существующие проблемы и станет фундаментом для дальнейшего развития средств профилирования в системе LuNA.

3 Проектирование архитектуры подсистемы профилирования

Архитектура программного обеспечения – это набор структурных элементов и их интерфейсов, при помощи которых компонуется система, вместе с их поведением, определяемым во взаимодействии между этими элементами [32]. Иногда под архитектурой программного обеспечения также подразумевают набор важных проектных решений о структурах программы и взаимодействиях между этими структурами. Проектные решения обеспечивают желаемый набор свойств, которые должна поддерживать система.

Для того, чтобы определить, какими свойствами должен обладать разрабатываемый инструмент профилирования, необходимо выделить требования, которым этот инструмент должен соответствовать.

Требования к программным продуктам можно разделить на две большие группы: функциональные и нефункциональные. *Функциональные требования* – это описание функций, которые должно выполнять программное обеспечение, чтобы удовлетворить потребности пользователей. Они описывают, что продукт должен делать, какие действия должны выполнять пользователи при взаимодействии с ним. *Нефункциональные требования* – это описание характеристик и свойств программного обеспечения, которые не связаны с его функциональностью, но важны для его эффективной работы и удовлетворения потребностей пользователей. К ним относятся, например, удобство использования системы, удобство сопровождения, расширяемость, безопасность.

3.1 Функциональные требования к подсистеме профилирования

Подсистема профилирования должна предоставлять пользователю следующие функции:

1. Возможность сбора трассы исполняемой фрагментированной программы.
2. Возможность сбора профиля исполняемой фрагментированной программы.
3. Возможность накопления информации об исполняемой фрагментированной программе во время ее исполнения для использования другими подсистемами в системе LuNA.
4. Возможность восстановления последовательности событий исполнения программы по трассировочным файлам.

При проектировании архитектуры подсистемы необходимо учитывать технические особенности исполнительной системы. В связи с этим, требуется обеспечить достоверность трассировочной и профилировочной информации для параллельных программ, использующих многопоточность и многопроцессность. Это относится к функциональным требованиям, так как напрямую влияет на то, будет ли способна система выполнять свои функции – собирать достоверную информацию об исполнении программ.

3.1.1 Сбор трассы исполняемой фрагментированной программы

Некоторым подсистемам, входящим в систему программирования LuNA, требуется подробная информация о поведении фрагментированной программы. Для корректной работы таких подсистем необходима возможность трассировки исполняемых программ.

Примером подсистемы, в которой используется трасса фрагментированной программы, является подсистема для распределения ресурсов фрагментированных программ, разработанная в работе Лямина А.С. «Разработка и реализация алгоритма распределения ресурсов фрагментированных программ на основе профилирования» [26]. В этой работе трасса обрабатывается специальной программой, которая определяет нагрузку на вычислительные узлы системы и при обнаружении дисбаланса корректирует распределение фрагментов вычислений по узлам. В процессе работы балансировщика нагрузки генерируется специальный файл с указаниями для последующих запусков данной фрагментированной программы. Для алгоритма обнаружения дисбаланса вычислительной нагрузки требуется подробная информация о событиях, связанных с выполнением фрагментов вычислений в исполнительной системе, которая содержится в трассе.

3.1.2 Сбор профиля исполняемой фрагментированной программы

Сбор профиля подразумевает возможность промежуточной обработки информации в процессе исполнения программы. Такую возможность нужно предусмотреть для случаев, в которых не требуется создание трассировочных файлов, но имеется необходимость в получении краткой статистики о поведении фрагментированной программы.

Примером подсистемы, использующей профиль LuNA-программы, является система для визуализации профиля фрагментированных программ, разработка которой описана в работе Голикова М.О. «Разработка и реализация средств для визуального анализа исполнения фрагментированных программ» [27]. Программное средство Evlog Profiler, описанное в работе, помимо графического представления информации, также предоставляет пользователю краткую статистику о программе, которая представлена на рисунке 4.

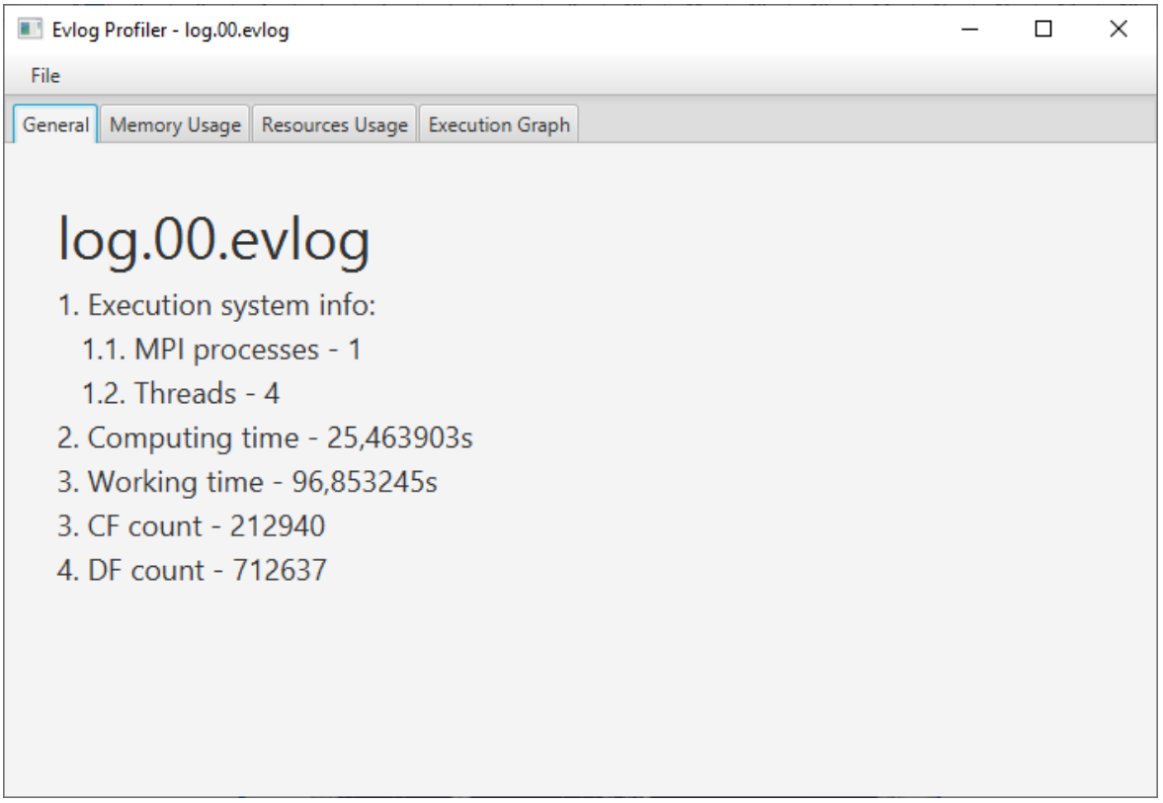

Рисунок 4 – Графический интерфейс статистической информации об исполнении фрагментированной программы

3.1.3 Накопление информации о программе во время ее исполнения

Информация об исполняемой фрагментированной программе может накапливаться в подсистеме профилирования и использоваться другими подсистемами в системе LuNA. Например, в процессе исполнения программы могут потребоваться данные о количестве потребляемой памяти или о текущей вычислительной нагрузке на узлах суперкомпьютера. Подсистема профилирования может собирать данную информацию, регистрируя и обрабатывая события, которые происходят в исполнительной системе.

Примером подсистемы, где нашла бы применение такая функциональность подсистемы профилирования, является модуль централизованной динамической балансировки вычислительной нагрузки в системе LuNA [33]. Вычислительная нагрузка в системе балансируется во время исполнения программы при помощи центрального узла, который получает информацию, необходимую для перераспределения нагрузки, от других узлов путем отправки сообщений. Для предоставления информации об актуальном состоянии исполнительной системы может использоваться динамическое профилирование.

3.1.4 Возможность восстановления последовательности событий исполнения программы по трассировочным файлам

Для некоторых задач требуется возможность восстановить точную последовательность событий в фрагментированной программе по трассировочным файлам.

Для этого каждое событие должно иметь временную метку, которая указывает, когда событие произошло в системе. Исполнительная система может вычислять фрагментированную программу на кластере, время на узлах кластера, возможно, не синхронизировано. Это может привести к различным проблемам при чтении и анализе трассировочных файлов. Например, рассмотрим ситуацию, в которой процесс отправил фрагмент данных процессу, расположенному на другом узле кластера. Из-за того, что время на узлах кластера может отличаться, возможна ситуация, в которой по временным меткам получение процессом фрагмента данных произошло раньше, чем отправка этого фрагмента. Такая ситуация невозможна, потому что пересылка фрагмента данных на другой узел требует ненулевого количества времени.

Описанная проблема приводит к тому, что в подсистеме возникает необходимость в использовании алгоритма синхронизации времени на узлах кластера. Это необходимо для того, чтобы по временным меткам в трассе можно было восстановить корректную последовательность событий исполнения программы.

3.2 Нефункциональные требования к подсистеме профилирования

Подсистема профилирования в системе программирования LuNA должна удовлетворять следующим нефункциональным требованиям:

1. Функциональность подсистемы должна быть расширяемой.
2. Накладные расходы на работу подсистемы должны быть минимизированы.
3. Подсистема должна иметь удобный и понятный пользовательский интерфейс.
4. Архитектура и функциональность подсистемы должны быть описаны в технической документации.

3.2.1 Расширяемость функциональности подсистемы

Расширяемость трассы означает возможность расширения списка регистрируемых событий, информация о которых сохраняется в трассировочные файлы, а также возможность расширения информации, собираемой о событиях.

Поскольку профиль отличается от трассы тем, что при профилировании информация не просто записывается, а обрабатывается каким-либо образом, расширяемость в контексте профилирования имеет другое значение. Для некоторых задач нам требуется возможность обрабатывать уже регистрируемые события другим способом для получения новых характеристик. Также расширение трассы и добавление новых событий потребует добавления в подсистему профилирования новых способов обработки и анализа информации.

3.2.2 Минимизация накладных расходов на работу подсистемы

Требуется, чтобы накладные расходы на работу подсистемы были минимальными. Это также подразумевает, что подсистема профилирования должна быть настраиваемой. Пользователь должен иметь возможность настроить систему и явно указать, какая именно информация об исполняемой программе и о состоянии исполнительной системы ему нужна. Это позволит пользователю не тратить ресурсы на получение информации, которая не требуется для данной конкретной задачи и не будет использована.

Более того, хотя сбор информации и не изменяет поведение фрагментированной программы, он влияет на процесс ее исполнения. Например, при записи трассировочных файлов может значительно увеличиться время исполнения программы. Такое влияние на процесс исполнения может ухудшить точность характеристик, полученных при профилировании, поэтому его необходимо минимизировать.

3.2.3 Пользовательский интерфейс подсистемы

Пользовательский интерфейс должен быть удобным и простым в использовании. Кроме этого, интерфейс должен быть понятным, чтобы у пользователя не возникало вопросов и сомнений о том, что ему необходимо сделать, чтобы собрать нужную информацию.

Для трассировки программ используется инструментирование кода. Инструментирование также увеличивает объем кода, может ухудшить читаемость кода. Поэтому необходимо спроектировать интерфейс таким образом, чтобы он был кратким, но при этом оставался понятным. Пользователь должен иметь возможность использовать подсистему, ничего не зная о ее внутреннем устройстве.

3.2.4 Документирование подсистемы

Подсистема профилирования должна быть описана в технической документации. Поскольку функциональность подсистемы профилирования будет расширяемой, это подразумевает, что подсистема должна быть документирована на двух уровнях: не только для пользователя, но и для разработчика.

Пользовательская документация предназначена для пользователя, то есть человека, который использует функции подсистемы профилирования, которые уже реализованы, и не вносит изменения в подсистему профилирования. Ознакомившись с документацией, пользователь должен получить представление о том, что ему необходимо сделать для того, чтобы использовать подсистему.

Подсистема должна иметь документацию для разработчика. Такая документация предназначена для программиста, который будет заниматься расширением функциональности подсистемы и модифицировать ее исходный код. Документация должна

содержать полное описание архитектуры подсистемы профилирования, а также основные принципы ее работы.

3.3 Описание предлагаемого решения

3.3.1 Использование событийно-ориентированного подхода

При исполнении фрагментированной программы в системе программирования LuNA происходят различные события, то есть существенные изменения состояния исполнительной системы. Информация о событиях должна быть передана в подсистему профилирования. Список типов событий, которые могут произойти в системе, является конечным. Архитектура подсистемы профилирования является *событийно-ориентированной*. Это означает, что система будет получать уведомления о происходящих в системе событиях, а также получать информацию об этих событиях.

Ответственность за сбор необходимых данных и передачу их в подсистему профилирования возлагается на исполнительную систему. Поскольку существуют различные версии системы LuNA, которые отличаются деталями реализации, такое требование поможет защитить подсистему профилирования от изменений, связанных с деталями реализации получения информации о событии и не имеющих отношения к функциональности. Это также сделает подсистему более универсальной, подходящей для различных версий системы LuNA.

В связи с этим, интерфейс подсистемы профилирования состоит из набора методов (в терминах языка C++) для передачи в подсистему информации о событии. Других методов пользовательский интерфейс подсистемы не предоставляет. Для каждого типа события в интерфейсе имеется специальный метод, который необходимо вызвать для того, чтобы уведомить подсистему профилирования о том, что в исполнительной системе произошло событие данного типа. Пусть для события типа **T** таким методом является **submit<T>** (от англ. *предоставить, отправить*). Необходимая информация о событии передается в качестве аргументов вызываемого метода. Для различных типов событий методы могут иметь различное число параметров, также могут отличаться типы параметров и их порядок.

3.3.2 Использование модульного подхода

Архитектура подсистемы профилирования является модульной, то есть подсистема профилирования представляет собой набор модулей. *Модуль* – это независимый компонент, функциональность которого ограничена одной зоной ответственности. Модули можно условно классифицировать по функциональности на две группы:

- модули, которые имеют состояние;
- модули, которые не имеют состояния.

Основное назначение модулей из первой группы – это обработка информации о событиях. Это позволяет подсистеме осуществлять сбор характеристик программы во время

ее исполнения. Модули, имеющие состояние, отражают текущее состояние исполнительной системы.

Согласно требованиям, в архитектуре также требуется предусмотреть возможность накопления информации об исполняемой фрагментированной программе во время ее исполнения. Данная информация будет использоваться другими подсистемами в системе LuNA. Для этого модуль должен иметь методы чтения (геттеры), которые будут предоставлять подсистемам доступ к состоянию модуля. Поскольку исполняемая в системе LuNA программа является многопоточной, методы, которые предоставляют доступ для чтения или модификации состояния модуля, являются критической секцией и требуют защиты от ошибок многопоточности.

Модули второй группы, которые не имеют состояния, не имеют отношения к состоянию исполнительной системы. Они выполняют действия, необходимые для функционирования подсистемы профилирования. Например, такой модуль может записывать информацию в поток вывода, приводить данные к нужному формату для записи в трассу или профиль.

Для обработки события у модуля необходимо вызвать специальный метод **notify** (от англ. *уведомлять, сообщать*) и передать необходимую модулю информацию в качестве аргументов. Под *вызовом модуля* здесь и далее имеется в виду вызов метода **notify** модуля. Для хранения и передачи информации между модулями в процессе обработки события в методе **submit<T>** создается и используется специальная структура данных. Информация, которая должна быть записана в трассу или в профиль, должна быть добавлена в эту структуру данных.

Вызов модулей для обработки информации о событии происходит в методе **submit<T>**. Каждый модуль имеет фиксированный список типов событий, для обработки которых он должен быть вызван, данный список должен быть непустым.

Модуль может иметь информационные зависимости от других модулей в случае, если ему необходима информация, получаемая в результате обработки события другим модулем. Примером такой ситуации может быть модуль форматирования данных, который должен получить информацию от некоторого набора модулей и преобразовать ее к нужному для записи формату. Поскольку вызов модулей для обработки события представляет собой последовательность вызовов метода **notify** модулей, необходимо, чтобы в зависимостях между модулями отсутствовали циклы. Иначе будет невозможно сформировать такую последовательность вызовов, в которой данные корректно обновляются и передаются в модули.

Разделение подсистемы на модули также позволит минимизировать накладные расходы двумя способами. Необходимо предоставить пользователю возможность выбирать необходимые модули при компиляции системы, а также включать или отключать использование некоторых модулей при запуске системы с помощью интерфейса командной строки.

3.3.3 Универсальный формат профиля

В методе **submit** для хранения информации, которая должна быть записана в трассу или в профиль, создается и используется специальная структура данных. В качестве такой структуры данных предлагается использовать ассоциативный контейнер, который хранит пары «ключ-значение», при этом ключи в контейнере должны быть уникальными. Ключом является название характеристики или параметра события. Например, для хранения временной метки ключом может быть значение **timestamp** (от англ. – временная метка).

В качестве основного универсального формата профиля предлагается использовать формат JSON. Данный формат также предназначен для хранения пар «ключ-значение», что обеспечивает совместимость с ассоциативным контейнером.

Тем не менее, в LuNA есть проекты, которые используют трассу, которая имеет определенный фиксированный формат. Запись трассы и/или профиля в таких форматах тоже должна поддерживаться подсистемой профилирования, что обеспечивается реализацией специальных модулей.

3.3.4 Алгоритм синхронизации времени

Для проведения синхронизации часов был выбран алгоритм Кристиана [34]. Алгоритм работает следующим образом. Сначала в системе выбирается машина, которая будет являться «сервером времени» (от англ. *time server*). Остальные машины в системе будут синхронизировать свое время со временем сервера: периодически каждая машина посылает серверу времени сообщение, запрашивая текущее время сервера. Машина, которая отправила серверу сообщение с запросом, называется *отправителем*. Сервер так быстро, как это возможно, отвечает сообщением, содержащим его текущее время, обозначим его T. Поскольку отправка и получение ответного сообщения сервера требуют ненулевого времени, которое также зависит от загрузки сети и может быть достаточно большим, отправитель измеряет интервал между отправлением запроса и получением ответа. Пусть T_0 – это время отправления запроса, а T_1 – это время получения ответа от временн**о**го сервера отправителем. В отсутствии какой-либо дополнительной информации наилучшим приближением времени прохождения сообщения будет $(T_1 - T_0)/2$. После получения ответа, чтобы получить

приблизительное текущее время сервера, значение, содержащееся в сообщении, следует увеличить на это число.

Для того, чтобы гарантировать, что никакая пара часов в системе не сможет разойтись более чем на δ, синхронизация часов должна производиться не реже, чем каждые $\delta/2\rho$ секунд. Константа ϱ определяется производителем процессора и известна под названием максимальной скорости дрейфа (англ. *maximum drift rate*) [34].

3.3.5 Документирование подсистемы профилирования

Подсистема должна содержать описание для каждого типа события, которое будет понятным для пользователя. Оно может описывать, например, при каких условиях происходит данное событие или что оно означает и какой смысл содержит.

Каждому типу события соответствует метод **submit**, вызываемый для передачи информации о событии в подсистему. Все методы, которые входят в пользовательский интерфейс подсистемы профилирования, обязаны иметь описание параметров: их количество, типы, назначение.

Все модули обязательно должны иметь документацию. Документация должна содержать описание задачи, для которой решения которой предназначен описываемый модуль. Также должны быть описаны количество и типы параметров метода **notify**, их назначение.

3.4 Анализ предлагаемого решения

3.4.1 Соответствие функциональным требованиям, предъявляемым к подсистеме профилирования

Предлагаемое решение соответствует функциональным требованиям, предъявляемым к подсистеме профилирования в системе программирования LuNA. Событийно-ориентированная архитектура подсистемы позволит системе получать уведомления о событиях и необходимую для обработки информацию. Представление подсистемы в виде набора модулей предоставляет возможность реализовать необходимую функциональность в виде модулей и в дальнейшем использовать их при обработке информации о событиях.

Функциональность модуля должна быть ограничена одной выполняемой модулем задачей. Тем не менее, никаких других ограничений на функции, которые обеспечивают модули, не ставится. Подсистема профилирования может иметь такие модули, которые реализуют конечные автоматы. Данная возможность является преимуществом, поскольку значительно расширяет область применения подсистемы профилирования в системе LuNA. Это позволяет сделать, например, модуль-отладчик, который следит за соответствием

происходящих событий некоторому конечному автомату, отражающему правильную работу системы.

Для решения проблемы синхронизации часов был выбран алгоритм, выполняющий синхронизацию времени в распределенной системе. Он может быть реализован в качестве отдельного модуля, который будет вызван сразу после начала исполнения программы для начальной синхронизации, а затем будет повторно вызываться в течение равных временных интервалов, длительность которых зависит от желаемой точности, для повторной проверки, что часы в системе синхронизированы.

Использование алгоритма синхронизации времени относится к необязательным функциональным требованиям, поскольку подразумевается, что пользователь сам вправе включать или отключать данную функцию при сборе трассы или профиля программы, в зависимости от того, требуется ли для решаемой задачи точный порядок событий в профиле на разных узлах.

3.4.2 Соответствие нефункциональным требованиям, предъявляемым к подсистеме профилирования

Использование модульной архитектуры предоставит большой ряд преимуществ с точки зрения нефункциональных требований к подсистеме профилирования.

Представление подсистемы в виде расширяемого набора модулей позволит сделать систему расширяемой. Добавление новой функциональности будет выполняться путем добавления в подсистему новых модулей. Модули также можно будет переиспользовать, поскольку в разных сценариях работы модули с одной и той же функциональностью могут быть использованы для различных задач.

Разделение подсистемы на модули также позволит минимизировать накладные расходы двумя методами:

- выбирать необходимые модули при компиляции системы;
- включать модули при запуске системы с помощью интерфейса командной строки.

Оба способа позволят пользователю самостоятельно выбирать, какие модули необходимо включить в подсистему для сбора необходимой информации, при этом избавляя его от необходимости тратить вычислительные ресурсы на сбор информации, которая пользователю не требуется в данный момент.

Возможность использования необходимых модулей также позволяет говорить о гибкости подсистемы профилирования. Под этим имеется в виду, что с помощью предлагаемой архитектуры подсистемы можно заменять модули перед запуском программы и получать различную функциональность. Например, заменив модуль для вывода информации

в файл модулем, который выводит информацию на экран или передает данные по сети, можно следить за характеристиками приложения в реальном времени.

Пользовательский интерфейс достаточно простой и удобный, поскольку включает в себя только методы для уведомления подсистемы о событии. Документация содержит всю необходимую информацию для того, чтобы использовать и/или модифицировать подсистему профилирования.

3.4.3 Соответствие принципам объектно-ориентированного дизайна

Как и сама система программирования LuNA, подсистема профилирования является объектно-ориентированной программой. Для объектно-ориентированных программ существуют принципы объектно-ориентированного дизайна. Часто для них используется аббревиатура SOLID. Принципы SOLID дают программисту возможность писать чистый, хорошо структурированный и легко обслуживаемый код. Соблюдению данных принципов при проектировании архитектуры было уделено внимание, так как они могут оказаться полезными для подсистемы, которая должна быть расширяемой и легко поддерживаемой.

Принцип единственной ответственности. Данный принцип означает, что каждый класс или модуль в программе должен нести ответственность только за одну часть функциональности этой программы. Подсистема профилирования соответствует данному принципу, поскольку она имеет модульную архитектуру, в которой каждый модуль выполняет единственную задачу, функциональность модуля ограничена одной зоной ответственности.

Принцип открытости/закрытости. Данный принцип означает, что классы и другие элементы должны быть открыты для расширения, но закрыты для модификации. Данный принцип также выполняется в подсистеме профилирования, поскольку подсистема является открытой для расширения при помощи добавления новых модулей, новых типов событий, новой собираемой информации о событиях. Тем не менее, функциональность, которая уже имеется в подсистеме, закрыта для модификации, изменение уже реализованных модулей не подразумевается.

Принцип разделения интерфейса. Данный принцип означает, что программные сущности не должны зависеть от методов, которые они не используют. Поскольку пользовательский интерфейс подсистемы был разработан специальным образом, согласно которому в него включаются только методы для передачи в подсистему информации о событиях, разделять интерфейсы не было необходимости. Интерфейсы модулей также спроектированы схожим образом: модули имеют публичный метод **notify**. Модули, которые имеют состояние, имеют методы для чтения данного состояния; список таких методов зависит от конкретного модуля. Значит, можно сделать вывод о том, что подсистема следует данному принципу.

Принцип подстановки Лисков. Данный принцип выполнен в подсистеме лишь частично из-за особенностей реализации подсистемы. Однако, выполнение данного принципа обеспечивается договоренностью о том, что модули имеют единый интерфейс и используют единственный публичный метод **notify**. Такая договоренность позволяет соблюдать данный принцип в архитектуре на уровне идеи, но не в реализации. То же применимо к **принципу инверсии зависимостей.** Данный принцип означает, что модули верхних уровней не должны зависеть от модулей нижних уровней. «Модулем верхнего уровня» в подсистеме является модуль, который содержит все методы пользовательского интерфейса. Данный модуль обращается к другим модулям подсистемы для обработки события – «модулям нижнего класса». Это частично выполнено при помощи договоренности об интерфейсе модулей, которые используют один метод **notify**. Однако, в реализации объединить модули при помощи одного интерфейса не удалось.

4 Реализация прототипа подсистемы профилирования

Реализация подсистемы профилирования без конкретной программы, для которой ее можно применить и протестировать, не имеет смысла. Поэтому было принято решение выбрать один из проектов, использующих трассировку или профилирование фрагментированных программ в системе LuNA, и реализовать прототип новой подсистемы профилирования, имеющий достаточную функциональность, чтобы он мог использоваться в проекте вместо ранее используемого инструмента. Таким образом, полученный прототип будет представлять собой MVP (от англ. *minimum viable product* – минимально жизнеспособный продукт), то есть продукт, который обладает минимальными, но достаточными для обеспечения потребностей первых пользователей функциями.

В связи с этим, в процессе реализации прототипа было принято решение обеспечить ту часть запланированной функциональности подсистемы профилирования, которая позволит практически применить работу в выбранном проекте и на основе тестирования сформулировать гипотезы о дальнейших направлениях развития подсистемы. Реализация функциональности, которая не была включена в процессе создания MVP, была временно отложена, поскольку ее реализация трудозатратна и не является критической для работы прототипа.

Реализация прототипа поможет проверить, что подсистема, имеющая предложенную архитектуру, способна обладать нужной функциональностью, позволит сравнить новую подсистему с уже существующими инструментами с точки зрения производительности и накладных расходов. Также это предоставит возможность практически проверить удобство интерфейса подсистемы профилирования и ее расширяемость – опытным путем узнать, что необходимо сделать разработчику и пользователю для того, чтобы реализовать и затем использовать предлагаемую архитектуру. Более того, это станет первым шагом к внедрению новой подсистемы профилирования в систему программирования LuNA.

Многие проекты в системе программирования LuNA основаны на посмертном профилировании, а оно в свою очередь требует выполнения трассировки исполняемой программы. Поэтому было принято решение реализовать прототип для одной из работ, которой необходима трассировка фрагментированных программ. Для реализации прототипа была выбрана работа Лямина А.С. «Разработка и реализация алгоритма распределения ресурсов фрагментированных программ на основе профилирования» [26], в которой требуется трассировка фрагментированной программы во время ее исполнения. После этого трасса анализируется при помощи алгоритма для балансировки вычислительной нагрузки в исполнительной системе, по результатам анализа строится корректирующее распределение фрагментов вычислений на вычислительные узлы системы.

Прототип разработан на языке C++. Руководство программиста по применению данной подсистемы приведено в Приложении А, описание программы представлено в Приложении Б.

4.1 Описание деталей реализации прототипа подсистемы профилирования

Исходный код подсистемы профилирования находится в пространстве имен (*namespace*) **pss** (от англ. ***p**rofiling **s**ub**s**ystem* – «подсистема профилирования»).

Список типов событий был зафиксирован в виде перечисляемого типа данных. Список включает в себя 22 типа события, из них 10 типов событий относятся к фрагментам вычислений, 7 – к фрагментам данных, 5 – к событиям системы программирования LuNA.

Для реализации пользовательского интерфейса подсистемы профилирования был использован класс **ProfSystem**. Данный класс написан с использованием шаблонов в языке C++ и может иметь частичные специализации. Для каждого типа события **T** класс **ProfSystem** имеет частичную специализацию **ProfSystem<T>**, которая содержит метод **submit**. Вызов данного метода пользователем представлен на листинге 1.

Листинг 1 – Вызов метода **submit** для события типа **T**

```
pss::ProfSystem<pss::Event::T>::submit(...);
```

В треугольных скобках указывается тип события, в круглых скобках указываются значения или переменные, которые передаются в метод в качестве аргументов. Поскольку для обработки различных типов событий требуется передавать в метод различное количество аргументов, важным требованием было переменное количество параметров в методах, а также различные типы параметров. Для решения данной проблемы было принято решение использовать вариативные шаблоны.

Другой возможный способ реализации данной функциональности, который был рассмотрен при разработке прототипа, заключается в использовании виртуальных функций. Применение виртуальных функций в реализации позволило бы использовать общий интерфейс для всех модулей подсистемы и, тем самым, соблюдать такие принципы объектно-ориентированного дизайна, как принцип подстановки Лисков и принцип инверсии зависимостей, рассмотренные в пункте 3.4.3. Однако, при использовании виртуальных функций не представляется возможным выполнение требования о различном количестве и типах параметров у методов интерфейса.

Также применение виртуальных функций плохо отразилось бы на производительности из-за значительного увеличения доли накладных расходов на работу подсистемы. Это связано с тем, что связывание для виртуальных функций происходит на этапе выполнения программы. В отличие от виртуальных функций, функции, написанные

при помощи шаблонов, создаются во время компиляции, вся информация, необходимая для вызова функции, будет доступна до выполнения программы. Значит, будет использоваться статическая привязка, которая приводит к более быстрому вызову программы, что соответствует требованию о минимизации накладных расходов.

Исполнение метода **submit** начинается с создания временной метки, которая указывает, когда произошло событие. Затем в теле метода создается ассоциативный контейнер **info**, который сопоставляет ключу некоторое значение (**std::map**). В данный контейнер записываются пары «ключ-значение», при этом ключи в контейнере должны быть уникальными. Данный контейнер заполняется информацией о событии, которая была передана в метод **submit**. Также данный контейнер может быть передан по ссылке в метод **notify** модулей, если они имеют информационные зависимости от других модулей.

Для записи информации в трассировочный файл в прототипе был реализован модуль **TraceWriter**. Данный модуль имеет метод **notify**, который принимает в качестве параметра ассоциативный контейнер с информацией о событии и выводит информацию в файл в заданном формате:

Номер процесса | Время | Информация о событии | Исходный файл:Строка.

4.2 Реализация алгоритма синхронизации часов

Для измерения времени в системе программирования LuNA был добавлен класс **Timer**, использующий для подсчета времени популярную библиотеку языка C++ – **std::chrono**. Таймер позволяет получить время в секундах, прошедшее с момента начала работы программы, и для этого сохраняет точку начала отсчета, то есть время начала исполнения программы.

Для синхронизации часов в прототипе был реализован алгоритм Кристиана. Роль сервера времени в системе программирования LuNA будет выполнять процесс с рангом 0, остальные процессы будут с ним синхронизироваться. Согласно алгоритму, если ранг процесса равен 0, то мы принимаем запросы от всех остальных процессов, каждому процессу в ответ на запрос отправляем текущее время процесса 0.

Остальные процессы являются отправителями. Перенос ответного сообщения с сервера требует ненулевого времени, поэтому каждый процесс измеряет свое время перед отправкой запроса (T_0) и после получения ответа от сервера (T_1). В реализации алгоритма для приближения времени прохождения сообщения используется значение $(T_1 - T_0)/2$.

Затем вычисляется значение $A = T_1 - T_A$, которое затем будет использоваться таймером при подсчете текущего времени. Данное значение передается в метод **sync** таймера и сохраняется для корректирования времени узла. Подсчет временной метки происходит по

следующей формуле: $t = T - T_0 - A$, где T – это текущее время процесса, T_0 – точка начала отсчета времени, A – отклонение от времени сервера. Если время процесса меньше, чем время сервера, значит, часы отстают, и отклонение будет со знаком минус. При подсчете времени оно будет прибавлено. И наоборот, если время процесса больше, то часы спешат относительно времени сервера, и отклонение нужно вычесть.

5 Тестирование прототипа подсистемы профилирования

Тестирование прототипа подсистемы проводилось на следующем оборудовании: ноутбук Lenovo Legion 5P 15IMH05, процессор Intel Core i5-10300H 2.50GHz, 4 ядра, 8 логических процессоров, 8 Гб DDR4 RAM, SSD 512 Гб.

Необходимо уточнить, что в предложенном решении каждый процесс исполнительной системы имеет отдельно исполняемый экземпляр кода подсистемы. Это означает, что при трассировке программы каждый процесс прототипа подсистемы профилирования генерирует трассировочный файл, в названии которого указывается номер процесса. В связи с этим тестирования прототипа подсистемы на одном процессе достаточно, и в тестировании прототипа на кластере нет необходимости.

5.1 Измерение доли накладных расходов на работу прототипа подсистемы

Идея тестирования заключается в том, чтобы применить для трассировки фрагментированной программы макрос, который использовался в системе LuNA ранее. Затем применить для трассировки этой же фрагментированной программы разработанный прототип подсистемы профилирования и сравнить полученные результаты между собой.

Для тестирования использовалась фрагментированная программа, реализующая алгоритм блочного умножения квадратных матриц. Этот тест целенаправленно нагружает подсистему трассировки, генерируя большое количество событий за короткий промежуток времени. Матрица является матрицей фрагментов данных, размер матрицы задается размером фрагмента и количеством фрагментов. Тестирование проводилось для матриц фрагментов данных размерностью 20×20, каждый фрагмент данных является подматрицей размера 200×200.

Результаты тестирования представлены в таблице 1. Исполнение фрагментированной программы в системе LuNA не является детерминированным, поэтому значения в таблице являются средними значениями для 20 запусков. Разброс по времени от среднего значения не превышал 25,7% для исполнения программы без трассировки и 10,2% для исполнения программы с трассировкой.

Таблица 1 – Результаты тестирования прототипа подсистемы профилирования

	Среднее значение, с	Минимальное значение, с	Максимальное значение, с
Время исполнения программы без трассировки	25,93022	23,1002	29,7716
Время исполнения программы с трассировкой при помощи макроса LOG	206,99535	200,012	211,537
Время исполнения программы с трассировкой при помощи прототипа новой подсистемы профилирования	66,017965	63,393	70,1329

Для вычисления доли накладных расходов использовалась следующая формула: $\text{Н} = \frac{\text{О}-\text{П}}{\text{П}} \times 100\%$, где О – это общее время исполнения программы с трассировкой, П – это время полезной нагрузки, которое равно времени исполнения программы без трассировки, Н – это доля накладных расходов на работу прототипа подсистемы в процентах.

По результатам тестирования видно, что в среднем доля накладных расходов на трассировку исполняемой программы составила 698,3% при использовании макроса LOG и 154,6% при использовании прототипа подсистемы. Из этого можно сделать вывод, что накладные расходы на трассировку фрагментированной программы уменьшились при использовании прототипа подсистемы. Следует, однако, учитывать, что доля накладных расходов зависит от степени фрагментации задачи. Если задача будет иметь более высокую степень фрагментации, то при исполнении программы будет происходить больше событий в исполнительной системе, что увеличит долю накладных расходов на работу подсистемы.

Такие результаты тестирования можно объяснить следующим образом. Реализация макроса LOG, которая использовалась для трассировки программы в данной работе, имеет некоторые проблемы с точки зрения производительности. При каждом вызове макроса выполняется функция **fopen**, которая открывает трассировочный файл и возвращает связанный с ним указатель, который в дальнейшем используется для записи информации в файл. Затем, после записи информации в файл, вызывается функция **fclose**, выполняющая закрытие файла. **fopen** и **fclose** – это функции из библиотеки языка C, которые внутри содержат как минимум один системный вызов. Системный вызов является более дорогой операцией, чем обычный вызов функции на уровне приложения, потому что требует переключения контекста, что влечет за собой большие накладные расходы.

5.2 Тестирование функциональности прототипа подсистемы

Для проведения тестирования использовалась программа, реализующая алгоритм балансировки вычислительной нагрузки, который анализирует трассу фрагментированной программы и строит корректирующее распределение фрагментов вычислений на вычислительные узлы системы [26]. Ранее в этой работе для трассировки фрагментированной программы использовался макрос LOG, рассмотренный в пункте 2.2.1 настоящей работы.

Идея тестирования заключается в том, чтобы запустить программу для балансировки нагрузки на трассе, сгенерированной при помощи прототипа новой подсистемы профилирования. Таким образом можно экспериментально проверить, что прототип реализует функциональность, которая необходима для трассировки фрагментированных программ, что в процессе трассировки генерируется корректный трассировочный файл, который можно использовать для балансировки вычислительной нагрузки в системе программирования LuNA.

Трассировочный файл, сгенерированный прототипом во время исполнения программы, идентичен трассировочному файлу, полученному с использованием макроса LOG, с точностью до временных меток событий. Информация о событиях в файлах совпадает, кроме временных меток. Инструменты вызываются последовательно, обработка информации о событии занимает некоторое количество времени, поэтому между вызовами присутствует небольшая задержка. Формат трассы, полученной при помощи разработанного прототипа, совпадает с форматом, который использовался ранее. Это означает, что трасса, генерируемая прототипом подсистемы профилирования, подходит для использования для балансировки вычислительной нагрузки в системе LuNA.

На рисунках 5 и 6 представлены результаты применения алгоритма балансировки вычислительной нагрузки, полученные для трассировочных файлов, сгенерированных в подразделе 5.1.

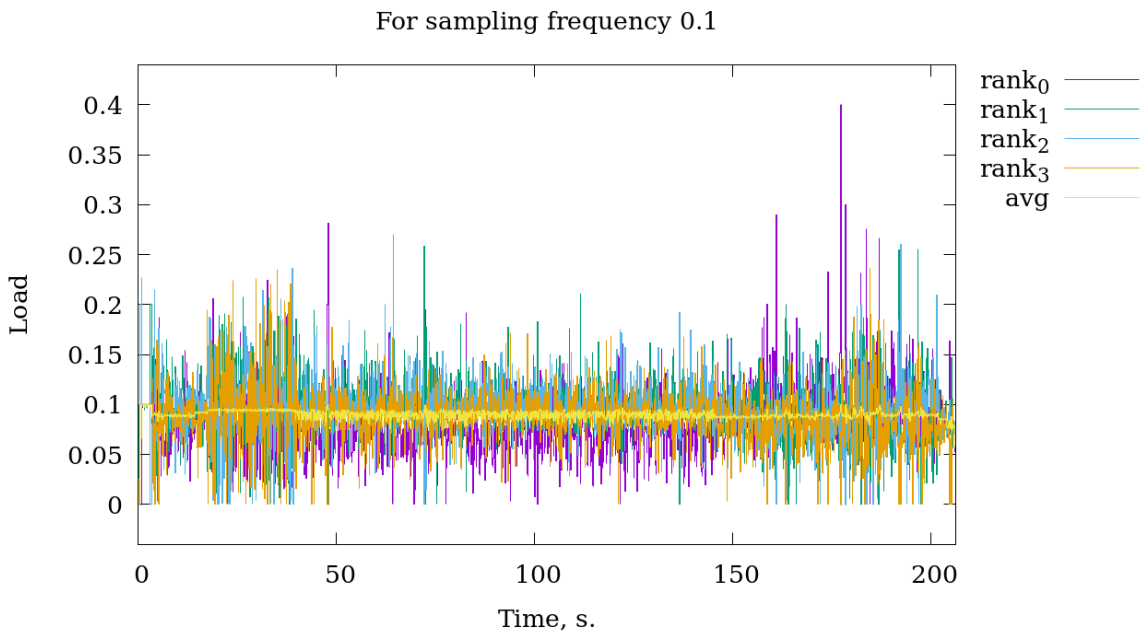

Рисунок 5 – График вычислительной нагрузки программы, трассировка которой была выполнена при помощи макроса LOG

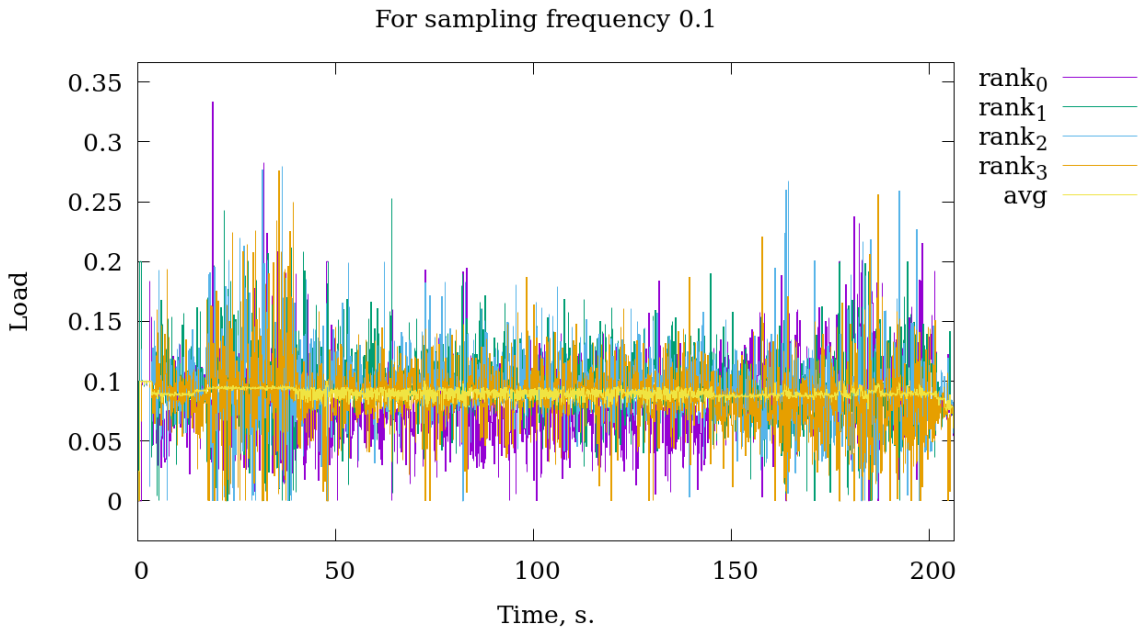

Рисунок 6 – График вычислительной нагрузки программы, трассировка которой была выполнена при помощи разработанного прототипа

Из рисунков видно, что трассу, полученную при помощи разработанного прототипа подсистемы, балансировщик нагрузки обрабатывает корректно. Во время работы алгоритм строит на основе трассы файл с указаниями для перебалансировки.

5.3 Итоги тестирования

При проведении тестирования предложенное решение продемонстрировало свою работоспособность. Разработанный прототип имеет функциональность, достаточную для

трассировки фрагментированных программ. Во время исполнения программы генерируются трассировочные файлы, которые содержат информацию о событиях в исполнительной системе. Формат трассы, полученной при помощи разработанного прототипа, совпадает с форматом, который использовался ранее, поэтому подходит для использования для балансировки вычислительной нагрузки в системе LuNA. Более того, использование прототипа позволило сократить накладные расходы на трассировку фрагментированных программ.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы был проведен обзор существующих средств профилирования параллельных программ. На основе анализа инструментов профилирования, представленных в обзоре, были сформулированы требования к подсистеме профилирования в системе фрагментированного программирования LuNA. В соответствии с выделенными требованиями разработана архитектура подсистемы профилирования в системе программирования LuNA, а также был реализован и протестирован прототип подсистемы, реализующий разработанную архитектуру.

Данная работа участвовала в Всероссийской летней XXXIX молодежной Школе-конференции по параллельному программированию с международным участием (4 – 15 июля 2022 г.) [35]. Также результаты работы были представлены на 61-й Международной научной студенческой конференции 17–26 апреля 2023 г. [36].

Защищаемые положения:

1. Архитектура подсистемы профилирования для системы фрагментированного программирования LuNA.
2. Экспериментальное исследование работоспособности прототипа подсистемы профилирования для системы фрагментированного программирования LuNA. Исследование показало, что использование прототипа позволило улучшить производительность исполнительной системы при трассировке фрагментированных программ.

В дальнейшем планируется развивать прототип подсистемы профилирования в системе программирования LuNA. Возможные направления развития:

1. Расширение функциональности прототипа для внедрения в другие проекты системы программирования LuNA, использующие профилирование.
2. Реализация разработанных способов уменьшения накладных расходов на работу подсистемы профилирования: возможности выбора необходимых модулей подсистемы профилирования при компиляции системы программирования LuNA, возможности включать модули при запуске системы программирования LuNA с помощью интерфейса командной строки.
3. Улучшение методов защиты критических секций в подсистеме профилирования от ошибок многопоточности.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной

литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлена с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

_____	_____
ФИО студента	*Подпись студента*

«____» _____ 20____ г.
(заполняется от руки)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. Издат. «Наука», Сибирское отделение, Новосибирск, 1988. – 129 с.
2. Малышкин В.Э. Технология фрагментированного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2012. №46 (305). С. 45 – 55.
3. V.E. Malyshkin, V.A. Perepelkin. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. // Proceedings of the 11th International Conference on Parallel Computing Technologies (PaCT-2011), LNCS, Vol. 6873. 2011. P. 53-61.
4. Sergey Kireev, Victor Malyshkin, Hamido Fujita. The LuNA Library of Parallel Numerical Fragmented Subroutines. // Proceedings of the 11th International Conference on Parallel Computing Technologies (PaCT-2011), LNCS, Vol. 6873. 2011. P. 290-301.
5. MPI: The Message Passing Interface [Электронный ресурс] // Лаборатория параллельных информационных технологий НИВЦ МГУ : официальный сайт – Режим доступа: https://parallel.ru/tech/tech_dev/mpi.html (дата обращения: 20.02.2022).
6. Гафаров Ф.М., Галимянов А.Ф. Параллельные вычисления: учеб. пособие / Ф.М. Гафаров, А.Ф. Галимянов. – Казань: Изд-во Казан. ун-та, 2018. – 149 с.
7. Чилингарова С.А. Методы оптимизации для динамических (just-in-time) компиляторов // Компьютерные инструменты в образовании. 2006. №2. URL: https://cyberleninka.ru/article/n/metody-optimizatsii-dlya-dinamicheskih-just-in-time-kompilyatorov (дата обращения: 10.05.2023).
8. MPICH: High-Performance Portable MPI [Электронный ресурс]. Режим доступа: https://www.mpich.org/about/overview/ (дата обращения: 17.04.2023).
9. S. Moore, D. Cronk, K. London, J. Dongarra. Review of performance analysis tools for MPI parallel programs // Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer, 2001. P. 241–248. Режим доступа: https://www.researchgate.net/publication/221597300_Review_of_Performance_Analysis_Tools_for_MPI_Parallel_Programs (дата обращения: 17.04.2023).
10. MPE, MultiProcessing Environment [Электронный ресурс] // Лаборатория параллельных информационных технологий НИВЦ МГУ : официальный сайт. Режим доступа: https://parallel.ru/docs/mpich/man/www4/MPE.html (дата обращения: 15.03.2022).

11. Intel-BSC Exascale Lab [Электронный ресурс] // Barcelona Supercomputing Center (Суперкомпьютерный центр Барселоны) : официальный сайт. Режим доступа: https://www.bsc.es/intel-bsc-exascale-lab (дата обращения: 17.04.2023).
12. Extrae [Электронный ресурс] // Barcelona Supercomputing Center (Суперкомпьютерный центр Барселоны) : официальный сайт. Режим доступа: https://tools.bsc.es/extrae (дата обращения: 12.04.2023).
13. Paraver [Электронный ресурс] // Barcelona Supercomputing Center (Суперкомпьютерный центр Барселоны) : официальный сайт. Режим доступа: https://tools.bsc.es/paraver (дата обращения: 12.04.2023).
14. D.A. Reed, P.C. Roth, R.A. Aydt, K.A. Shields, L.F. Tavera, R.J. Noe, and B.W. Schwartz. Scalable performance analysis: The Pablo Performance Analysis Environment. // Proceedings of the Scalable Parallel Libraries Conference, 1993. P. 104–113. Режим доступа: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=6e0039d62431ec95136f738c5020f6e3d3711168 – DOI: 10.1109/SPLC.1993.365577.
15. Филамофитский М. Средства анализа производительности параллельных приложений [Электронный ресурс] // Лаборатория параллельных информационных технологий НИВЦ МГУ : официальный сайт – Режим доступа: https://parallel.ru/tech/perf/profilers.html (дата обращения: 12.04.2022).
16. VampirTrace [Электронный ресурс] // Center for Information Services and High Performance Computing (ZIH) (Центр информационных услуг и высокопроизводительных вычислений) : официальный сайт Дрезденского технического университета. Режим доступа: https://tu-dresden.de/zih/forschung/projekte/vampirtrace (дата обращения: 10.05.2023).
17. The Vampir: History & Development [Электронный ресурс] // The Vampir Tool : официальный сайт. Режим доступа: https://vampir.eu/history (дата обращения: 10.05.2023).
18. W. Nagel, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. Supercomputer, 12(1):69–80, 1996. Режим доступа: https://juser.fz-juelich.de/record/189233/files/ib-9528.pdf (дата обращения: 10.05.2023).
19. Intel Trace Analyzer and Collector [Электронный ресурс] // Официальный сайт корпорации Intel. Режим доступа: https://www.intel.com/content/www/us/en/developer/tools/oneapi/trace-analyzer.html (дата обращения: 10.05.2023).
20. Scalasca [Электронный ресурс] // Scalasca : официальный сайт. Режим доступа: http://www.scalasca.org (дата обращения: 10.05.2023).

21. B. Mohr, F. Wolf. KOJAK – A Tool Set for Automatic Performance Analysis of Parallel Programs // European Conference on Parallel Processing, LNCS 2790, Springer-Verlag Berlin Heidelberg, 2003. P. 1301-1304. Режим доступа: https://www.researchgate.net/publication/225142840_KOJAK_-_A_Tool_Set_for_Automatic_Performance_Analysis_of_Parallel_Programs (дата обращения: 17.04.2023) – DOI: 10.1007/978-3-540-45209-6_177
22. M. Geimer, F. Wolf, B. Wylie, E. Abraham, D. Becker, B. Mohr. The Scalasca performance toolset architecture // Concurrency and Computation: Practice and Experience, 2010. P. 702-719. Режим доступа: https://apps.fz-juelich.de/jsc-pubsystem/aigaion/attachments/sthec08-special.pdf-c565f219e0f9e1814e48c543dfe71039.pdf (дата обращения: 17.04.2023).
23. S. S. Shende, A. D. Malony. The TAU Parallel Performance System // International Journal of High Performance Computing Applications, 20(2), 2006. P. 287–331. Режим доступа: https://www.researchgate.net/publication/220457208_The_Tau_Parallel_Performance_System (дата обращения: 12.04.2023).
24. Чайковский Д. С., Гулевич Н. А. Анализ современных средств профилирования параллельных программ // Вестник СГТУ. 2014. №1 (76). Режим доступа: https://cyberleninka.ru/article/n/analiz-sovremennyh-sredstv-profilirovaniya-parallelnyh-programm (дата обращения: 12.04.2023).
25. HPCToolkit Home [Электронный ресурс] // HPCToolkit : официальный сайт. Режим доступа: http://hpctoolkit.org/ (дата обращения: 12.04.2023).
26. Лямин А.С. Разработка и реализация алгоритма распределения ресурсов фрагментированных программ на основе профилирования // Выпускная работа бакалавра, НГУ, Факультет информационных технологий. – 2021.
27. Голиков М.О. Разработка и реализация средств для визуального анализа исполнения фрагментированных программ // Выпускная работа бакалавра, НГУ, Факультет информационных технологий. – 2021.
28. Мичуров М. А. Разработка и реализация программных средств автоматизированного обнаружения семантических ошибок во фрагментированных программах для системы LuNA // Выпускная работа бакалавра, НГУ, Факультет информационных технологий. – 2022.
29. Саяпин М.П. Разработка и реализация алгоритмов динамической балансировки вычислительной нагрузки для подсистемы воспроизведения трасс в системе LuNA. Выпускная работа бакалавра, НГТУ, Факультет прикладной математики и информатики. – 2022.

30. Логирование событий для профилирования системы LuNA [Электронный ресурс] // LuNA Wiki : документация системы программирования LuNA. Режим доступа: https://gitlab.ssd.sscc.ru/luna/luna5/-/wikis/luna_evlog_doc (дата обращения: 24.04.2023).
31. Анализаторы профиля исполнения LuNA-программы [Электронный ресурс] // LuNA Wiki : документация системы программирования LuNA. Режим доступа: https://gitlab.ssd.sscc.ru/luna/luna5/-/wikis/luna-evlog-analyzer (дата обращения: 24.04.2023).
32. Назаров С.В. Архитектура и проектирование программных систем : монография / С.В. Назаров – 2-е изд., перераб. и доп. – М. : ИНФРА-М., 2016. – 374 с.
33. Мустафин Д.Э. Реализация централизованного подхода к динамической балансировке вычислительной нагрузки в системе фрагментированного программирования LuNA и его сравнение с децентрализованным подходом. Выпускная работа бакалавра, НГУ, Факультет информационных технологий. – 2022.
34. Таненбаум Э., Стеен М. ван., Распределенные системы. Принципы и парадигмы / Серия «Классика computer science». - СПб.: Питер. - 2003. - С. 877.
35. Абрамушкина Е.С. Проектирование подсистемы профилирования для системы фрагментированного программирования LuNA // Всероссийская летняя XXXVII молодежная Школа-конференция по параллельному программированию с международным участием (г. Новосибирск, 5 – 16.07.2021 г.). Всероссийская летняя XXXIX молодежная Школа-конференция по параллельному программированию с международным участием (г. Новосибирск, 4 – 15.07.2022 г.): Тез. докл. / отв. ред. д-р техн. наук, проф. В. Э. Малышкин. Новосиб. гос. унт. – Новосибирск : ИПЦ НГУ (принято в печать).
36. Абрамушкина Е.С. Разработка подсистемы профилирования для системы LuNA // Параллельные вычисления : Материалы 61-й Междунар. науч. студ. конф. 17–26 апреля 2023 г. / Новосиб. гос. ун-т. – Новосибирск : ИПЦ НГУ (принято в печать).

ПРИЛОЖЕНИЕ А

ПОДСИСТЕМА ПРОФИЛИРОВАНИЯ
ДЛЯ СИСТЕМЫ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ
«LUNA»

Руководство программиста

Листов 6

Новосибирск 2023

СОДЕРЖАНИЕ

Аннотация..... 49

1. Назначение программы..... 50

1.1. Функциональное назначение программы..... 50

1.2. Эксплуатационное назначение программы..... 50

1.3. Состав функций..... 50

2. Условия выполнения программы..... 51

2.1. Минимальный состав аппаратных средств..... 51

2.2. Минимальный состав программных средств..... 51

2.3. Требования к персоналу..... 51

3. Выполнение программы..... 52

3.1. Загрузка и запуск программы..... 52

3.2. Выполнение программы..... 52

3.3. Завершение работы программы..... 52

АННОТАЦИЯ

В данном документе приведено руководство программиста для подсистемы профилирования для системы LuNA. Программа написана на языке C++.

Оформление программного документа «Руководство программиста» произведено по требованиям ГОСТ 19.504-79 «ЕСПД. Руководство программиста» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением № 1)».

1. Назначение программы

1.1. Функциональное назначение программы

Подсистема профилирования предназначена для сбора информации об исполняемых фрагментированных параллельных программах в системе программирования LuNA.

1.2. Эксплуатационное назначение программы

Программный модуль предназначен для сбора информации об исполняемых фрагментированных параллельных программах в системе LuNA. Поскольку профилировочная информация в системе программирования LuNA используется для анализа производительности системы и исследования методов ее повышения, программный модуль способен облегчить достижение высокой производительности системы программирования LuNA.

1.3. Состав функций

- Сбор и промежуточная обработка внутренней информации о системных событиях, происходящих с фрагментами вычислений.
- Сбор и промежуточная обработка внутренней информации о системных событиях, происходящих с фрагментами данных.
- Сбор информации о потреблении ресурсов исполнительной системой при исполнении фрагментированной программы.
- Возможность сохранения собранной информации в файл.

2. Условия выполнения программы

2.1. Минимальный состав аппаратных средств

Программа предусмотрена для использования на персональном компьютере или вычислительном кластере. Устройство, на котором запускается программа, должно иметь:

- оперативную память объемом 4 Гб или выше;
- процессор с тактовой частотой 1 ГГц или выше;
- жесткий диск объемом 32 Гб или выше;
- монитор;
- клавиатуру.

2.2. Минимальный состав программных средств

Системные программные средства, используемые подсистемой профилирования фрагментированных программ, должны быть представлены:

- Операционной системой, поддерживающей интерпретатор языка Python версии 3.8 и выше и компилятор GNU C++.
- Интерпретатор языка Python версии 3.8 и выше.
- Компилятор языка GNU C++.

2.3. Требования к персоналу

Конечный пользователь программы (оператор) должен обладать практическими навыками использования интерфейса командной строки и опытом использования системы LuNA.

3. Выполнение программы

3.1. Загрузка и запуск программы

Для работы подсистемы предварительно необходимо провести инструментирование кода, то есть дополнить исходный код программы вызовами инструмента для профилирования. Запуск подсистемы профилирования происходит одновременно с запуском системы LuNA, так как является его составной частью.

3.2. Выполнение программы

После запуска подсистема профилирования на основе данных об исполнении программы генерирует файл, который в зависимости от выбранных параметров содержит трассу или профиль исполняемой программы.

3.3. Завершение работы программы

Завершение программы происходит автоматически после окончания вычислений. Программу можно прервать в любой момент, используя сочетание клавиш Ctrl+C, это сочетание вызовет сигнал SIGINT.

ПРИЛОЖЕНИЕ Б

ПОДСИСТЕМА ПРОФИЛИРОВАНИЯ
ДЛЯ СИСТЕМЫ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ
«LUNA»

Описание программы

Листов 12

Новосибирск 2023

СОДЕРЖАНИЕ

Аннотация..... 55

1. Общие сведения..... 56

1.1. Обозначение и наименование программы..... 56

1.2. Программное обеспечение, необходимое для функционирования программы..... 56

1.3. Языки программирования..... 56

2. Функциональное назначение..... 57

2.1. Назначение программы..... 57

2.2. Сведения о функциональных ограничениях на применение..... 57

3. Описание логической структуры..... 58

3.1. Алгоритм программы..... 58

3.2. Структура программы..... 58

3.3. Связи между составными частями программы..... 58

3.4. Связи программы с другими программами..... 58

4. Используемые технические средства..... 59

5. Вызов и загрузка..... 60

6. Входные данные..... 61

7. Выходные данные..... 62

8. Лист регистрационных изменений..... 63

АННОТАЦИЯ

В данном документе приведено описание программы подсистемы профилирования для системы LuNA. Программа написана на языке C++. Средство разработки – интегрированная среда разработки Visual Studio Code от компании Microsoft.

Оформление программного документа «Описание программы» произведено по требованиям ГОСТ 19.402-78 «ЕСПД. Описание программы» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением № 1)».

1. Общие сведения

1.1. Обозначение и наименование программы

Полное наименование программного модуля – «Подсистема профилирования для системы фрагментированного программирования LuNA». Краткое название – «Подсистема профилирования LuNA».

1.2. Программное обеспечение, необходимое для функционирования программы

- Интерпретатор языка Python версии 3.8 и выше.
- Компилятор языка C++ GCC.

1.3. Языки программирования

Программа написана на языке C++. Средство разработки – интегрированная среда разработки Visual Studio Code от компании Microsoft.

2. Функциональное назначение

2.1. Назначение программы

Программный модуль предназначен для сбора информации об исполняемых фрагментированных параллельных программах в системе LuNA. Поскольку профилировочная информация в системе программирования LuNA используется для анализа производительности системы и исследования методов ее повышения, программный модуль способен облегчить достижение высокой производительности системы программирования LuNA.

2.2. Сведения о функциональных ограничениях на применение

Программа не предназначена для запуска на платформах, не поддерживающих C++ 17 или Python 3.8.

3. Описание логической структуры

3.1. Алгоритм программы

Программа встроена в один из основных модулей системы LuNA: исполнительную систему. Функциональность, встроенная в исполнительную систему, сохраняет данные о вычисляющем устройстве и производит подстановку значений, если необходимо.

3.2. Структура программы

Программа является расширением одного из двух основных модулей системы программирования LuNA.

Таблица 2 – Описание структуры подсистемы профилирования

Модуль	Описание
ProfSystem	Модуль, реализующий пользовательский интерфейс подсистемы профилирования
TraceWriter	Модуль, реализующий запись информации в трассировочный файл
Timer	Предназначен для измерения времени в системе программирования LuNA.

3.3. Связи между составными частями программы

Связь между составными частями выполняется следующим образом: один из модулей, который предоставляет пользовательский интерфейс, имеет специальную функцию, в которой вызываются другие модули для обработки информации о событии.

3.4. Связи программы с другими программами

Для запуска программы необходимы интерпретатор языка Python версии 3.8 и выше, а также компилятор языка GNU C++.

4. Используемые технические средства

Программа эксплуатируется на персональном компьютере или вычислительном кластере. Работа выполняется в консоли.

Устройство, на котором запускается программа, должно иметь:

- оперативную память объемом 4 Гб или выше;
- процессор с тактовой частотой 1 ГГц или выше;
- жесткий диск объемом 32 Гб или выше;
- монитор;
- клавиатура.

5. Вызов и загрузка

Для работы подсистемы предварительно необходимо провести инструментирование кода, то есть дополнить исходный код программы вызовами инструмента для профилирования. Запуск подсистемы профилирования происходит одновременно с запуском системы LuNA, так как является его составной частью.

6. Входные данные

Подсистема профилирования должна эксплуатироваться только в составе системы программирования LuNA. Пользователь задает список событий фрагментированной программы, информацию о которых необходимо собрать в процессе исполнения программы.

7. Выходные данные

Пользователь может выбрать два режима работы: сбор трассы и/или профиля. Для получения выходных данных пользователь может использовать вывод собранной информации в файл.

8. Лист регистрационных изменений

Таблица 3 – Лист регистрационных изменений

Лист регистрационных изменений									
Номера листов (страниц)					Всего листов (страниц) в докум.	№ доку-мента	Входящий № сопроводит. докум. и дата	Подп.	Дата
Изм.	измене-нных	замене-нных	но-вых	аннулиро-ванных					